Raccourci

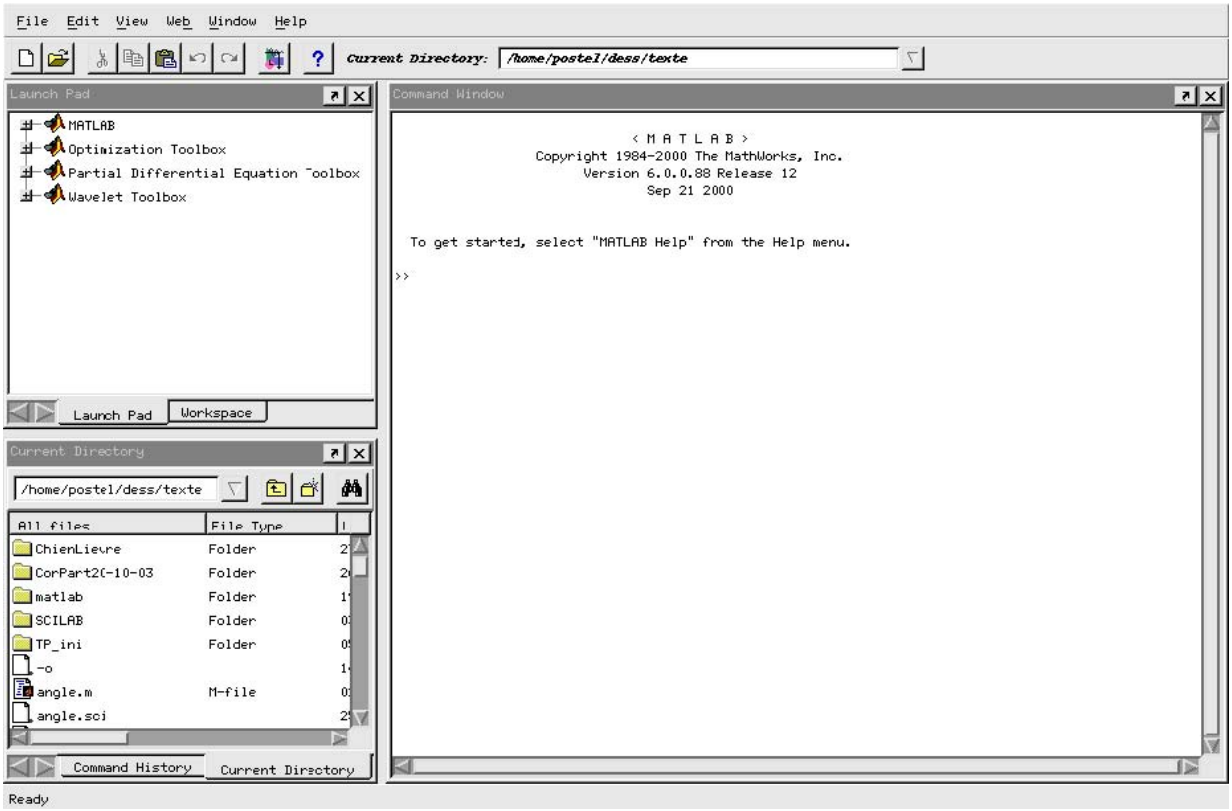
pour apprendre

MATLAB

Introduction

MATLAB est un logiciel commercial de calcul interactif. Il permet de réaliser des simulations numériques basées sur des algorithmes d'analyse numérique. Il peut donc être utilisé pour la résolution approchée d'équations différentielles, d'équations aux dérivées partielles ou de systèmes linéaires, de modélisation de file d'attente, ... L'objectif de ces séances Matlab est double: la connaissance de ce logiciel est en soi indispensable parce qu'il est de plus en plus utilisé dans l'industrie pour développer des prototypes de logiciels et tester de nouveaux algorithmes. Signalons au passage que des logiciels en shareware/freeware émulent Matlab de manière de plus en plus satisfaisante. Leur utilisation permet de palier l'inconvénient principal du coût de la licence. Vous pouvez par exemple télécharger Scilab gratuitement sur le site Internet de l'INRIA.

Interface de Matlab :



Syntaxe du langage

Un script Matlab est composé d'une suite d'instructions, toutes séparées par une virgule (ou de manière équivalente, un passage à la ligne) ou un point virgule. La différence entre ces deux types de séparation est liée à l'affichage ou non du résultat à l'écran (seulement effectué dans le premier cas). Comme tout langage, Matlab possède aussi un certain nombre d'instructions syntaxiques (boucles simples, conditionnelles, ...) et de commandes élémentaires (lecture, écriture,...). Dès que le calcul à effectuer implique un enchaînement de commandes un peu compliqué, il vaut mieux écrire ces dernières dans un fichier. Par convention un fichier contenant des commandes Matlab porte un nom avec le suffixe .m et s'appelle pour cette raison un m-file ou encore script. On utilisera l'éditeur intégré au logiciel qui se lance à partir de la fenêtre de commande en cliquant sur les icônes new m-file ou open file dans la barre de menu ou en lançant la commande edit. Une fois le fichier enregistré sous un nom valide, on peut exécuter les commandes qu'il contient en tapant son nom - sans le suffixe .m - dans la fenêtre de commande. Si vous avez ouvert l'éditeur comme indiqué, à partir de la fenêtre de commande, les m-file seront créés dans le répertoire courant, accessible depuis cette fenêtre, et vous n'aurez pas de problème d'accès. Si vous voulez exécuter des scripts qui se trouvent ailleurs dans l'arborescence des fichiers, vous aurez éventuellement à modifier le Path en cliquant sur le menu file -> SetPath ou bien en changeant de répertoire de travail (cliquer sur l'onglet current directory).

Utilisation de l'aide en ligne

L'apprentissage du bon maniement de l'aide (en ligne ou papier) du logiciel est indispensable: étant donné le très grand nombre d'instructions utilisables, il est hors de question de pouvoir mémoriser chacune d'elles avec sa syntaxe correspondante. À noter également que cette aide est uniquement disponible en langue anglaise (ce qui nécessite le cas échéant l'apprentissage complémentaire de quelques rudiments de cette langue...). L'aide en ligne permet de retrouver toutes les informations utiles: soit en tapant help + nom de la commande, soit en allant dans le menu Help->MATLAB help, une nouvelle fenêtre s'ouvre partagée en deux. A gauche, on peut en cliquant sur le signet supérieur correspondant, activer

- Contents. La table des matières de l'aide en ligne. En cliquant sur un chapitre de l'arborescence, son contenu est affiché dans la fenêtre de droite.
- Index. L'index de l'aide en ligne, qui répertorie toutes les commandes Matlab et permet d'accéder à leur mode d'emploi spécifique. On peut taper le nom de la commande - si on la connaît !- dans la fenêtre de saisie Search index for:, ou bien la rechercher dans la liste alphabétique proposée. Pour avancer dans l'alphabet taper la première lettre de la commande recherchée dans la fenêtre de saisie.
- Search. Un moteur de recherche dans l'index

- Favorites. La liste des chapitres de l'aide stockés dans les favoris. Pour ajouter un chapitre dans cette liste, se positionner dessus à partir de Contents et cliquer sur le bouton de droite de la souris.

Une aide très importante se présente sous la forme de programmes de démonstration auxquels on peut accéder à partir de Contents -> Begin Here puis en cliquant sur demos dans la sous-fenêtre de droite, ou bien en tapant demos directement dans la fenêtre de commande.

Fenêtre de commande

Les caractères >> en début de ligne constituent le prompt de Matlab. C'est après eux que vous pouvez taper des commandes qui seront exécutées par le logiciel après avoir tapé sur la touche entrée. Le résultat de l'exécution s'inscrit alors dans la fenêtre ou est représenté graphiquement dans une nouvelle fenêtre spécifique (avec possibilité de zoom, d'impression, etc...). Pour rentrer une suite complexe d'instructions (on parle aussi d'un script), on les tape au préalable dans un fichier en utilisant l'éditeur intégré. Une fois le script enregistré, on peut l'exécuter en tapant son nom dans la fenêtre Matlab. L'historique des instructions entrées depuis le début de la session sur la ligne de commande est accessible par pression successive de la touche ↑. Enfin, pour effacer les données en mémoire (par exemple avant d'exécuter un nouveau calcul), il suffit d'utiliser la commande clear.

Si vous ne désirez pas voir le résultat d'une commande immédiatement, il faut la faire suivre d'un point virgule. La commande sera exécutée, mais son résultat n'apparaîtra pas à l'écran.

Enfin si vous voulez que Matlab ignore complètement ce que vous tapez (ce qui sera utile dans l'écriture des scripts pour y mettre des commentaires améliorant leur lisibilité), il faut le faire précéder du caractère %.

Plusieurs types de données sont disponibles dans Matlab. Les types traditionnels que l'on retrouve dans tous les langages de programmation: les types numériques (single, double, int8, etc...), caractères char, les tableaux de réels, et les tableaux creux sparse, et les types composés cell, structure ainsi que les types définis par l'utilisateur, comme les fonctions inline. Le type de donnée privilégié sous Matlab est les **tableaux** à une ou deux dimensions, qui correspondent aux vecteurs et matrices utilisés en Mathématiques et qui sont aussi utilisés pour la représentation graphique.

Vecteurs

Pour définir un vecteur la syntaxe est une des suivantes:

```
»v=[2;3;7]              %vecteur colonne composantes réelles
v=
   2.0000
```

```
   3.0000
   7.0000
»v=[2,-3+i,7]   %vecteur ligne composantes complexes, i^2 =-1
v =
   2.0000           -3.0000 + 1.0000i   7.0000
»v'                          % vecteur transconjugué
ans =
   2.0000
  -3.0000 - 1.0000i
   7.0000
»v.'                         % vecteur transposé
ans =
   2.0000
  -3.0000 + 1.0000i
   7.0000
»w=[-3;-3-i;2]               % vecteur colonne
w =
  -3.0000
  -3.0000 - 1.0000i
   2.0000
»v+w'                        % somme de deux vecteurs
ans =
  -1.0000           -6.0000 + 2.0000i   9.0000
»v*w                         % produit scalaire euclidien
 ans  =
    18.
»w'.*v    % produit des composantes terme à terme
ans =
  -6.0000            8.0000 - 6.0000i  14.0000
»w'.*v    % division des composantes terme à terme
»w.^3     % mise à la puissance 3 de chaque composante.
```

Les composantes sont séparées par des blancs (dangereux) ou de préférence par des virgules pour les vecteurs lignes et par des points-virgules pour les vecteurs colonnes. Des messages erreurs sont affichés si une opération impossible est tentée (par exemple l'addition de vecteurs de longueurs différentes).

Et pour aller plus vite...

```
»v=1:5.1:23           % vecteur a increment constant
v =
    1.0000    6.1000   11.2000   16.3000   21.4000
»ones(size(v)) % vecteur de meme taille que v et contenant des 1
 ans  =
   1  1  1  1  1
```

```
»ones(1,4) % vecteur ligne 4 composantes egales a 1
 ans  =
   1    1    1   1
»3*ones(1,5)  % vecteur ligne 5 composantes egales a 3
 ans  =
   3    3    3    3    3
»zeros(3,1)  % vecteur colonne 3 composantes nulles
 ans  =
    0
    0
    0
```

Matrices

Les matrices suivent la même syntaxe que les vecteurs. Les composantes des lignes sont séparées par des virgules et chaque ligne est séparée de l'autre par un point virgule.

```
»% une manière de définir une matrice 3 x 3:
»A=[1,2,3;0,0,atan(1);5,9,-1];

»% une autre syntaxe pour faire la même chose
»A=[1 2 3
»     0 0 atan(1)
»     5 9 -1]
 A  =
   1.    2.    3.
   0.    0.     .7853982
   5.    9.  - 1.

»% à ne pas confondre avec ce groupe d'instructions
»A=[1 2 3...
»0 0 atan(1)...
»5 9 -1]
 A  =
   1.    2.    30.    0.    0.7853982    5.    9.  - 1.


»v=1:5;W=v'*v          % multiplication de matrices
W =
     1     2     3     4     5
     2     4     6     8    10
     3     6     9    12    15
     4     8    12    16    20
     5    10    15    20    25
»W(1,:)              %extraction de  la première ligne
```

```
 ans  =
   1.    2.    3.    4.    5.
»A=eye(3,3)     % Matrice identité
A =
    1    0    0
    0    1    0
    0    0    1
»B=toeplitz([2,1,0,0])   % une matrice de Toeplitz
B =
    2    1    0    0
    1    2    1    0
    0    1    2    1
    0    0    1    2
```

Le tableau 1 résume les principales fonctions affectant ou effectuant des opérations sur des matrices. Noter que les fonctions scalaires courantes, (sin, exp, ...) peuvent aussi s'appliquer à des matrices, composante par composante, comme dans l'exemple suivant

```
»u=[0:1:4]
u =
    0    1    2    3    4
»v=sin(u)
v =
       0    0.8415    0.9093    0.1411   -0.7568
```

Tableau 1: Principales opérations sur les matrices.

Fonction	Description
ones(i,j)	crée un tableau de i lignes j colonnes contenant des 1
zeros(i,j)	crée un tableau de i lignes j colonnes contenant des 0
eye(i,j)	crée un tableau de i lignes j colonnes avec des 1 sur la diagonale principale et 0 ailleurs
toeplitz(u)	crée une matrice de Toeplitz symétrique dont la première ligne est le vecteur u
diag(u)	crée une matrice carrée avec le vecteur u sur la diagonale et 0 ailleurs
diag(U)	extrait la diagonale de la matrice U
triu(A)	renvoie la partie supérieure de A
tril(A)	renvoie la partie inférieure de A
linspace(a,b,n)	crée un vecteur de n composantes uniformément réparties de a à b
cond(A)	conditionnement d'une matrice (norme euclidienne)

det(A)	déterminant d'une matrice
rank(A)	rang d'une matrice
inv(A)	inverse d'une matrice
pinv(A)	pseudo inverse d'une matrice
svd(A)	valeurs singulières d'une matrice
norm(A)	norme matricielle ou vectorielle
u'	prend le transposé de u
u*v	multiplication matricielle
u+v	addition matricielle
u-v	soustraction matricielle
u.* v	multiplication des tableaux u et v terme à terme
u./v	division du tableau u par le tableau v terme à terme
find(C(A))	indices des composantes du tableau A vérifiant la condition C(A)

Fonctions

Nous allons maintenant détailler quelques règles de programmation qui pour la plupart ne devraient pas surprendre outre mesure si on connaît déjà un autre langage (C ou Fortran).

Les fonctions sont des enchaînements de commandes Matlab regroupées sous un nom de fonction permettant de commander leur exécution.

On peut mettre dans une fonction un groupe de commandes destiné à être exécuté plusieurs fois au cours du calcul avec éventuellement des valeurs de paramètres différents. La fonction peut aussi être chargée de réaliser un calcul avec un certain algorithme, qui pourra être remplacé éventuellement par un autre plus rapide ou plus précis, en changeant simplement le nom de la fonction dans le programme appelant. Enfin, dès que le programme est un peu long et compliqué, il est souhaitable de le découper en fonctions, correspondant à des étapes pour améliorer la lisibilité et la compréhension de l'algorithme.

Les fonctions peuvent être définies inline c'est à dire dans le corps du programme appelant ou bien dans des fichiers externes indépendants m-file

m-files functions

Dès que la fonction nécessite plusieurs instructions, il vaut mieux la définir dans un fichier à part à l'aide de l'éditeur de texte.

De manière générale, la syntaxe de définition d'une fonction externe est

function [y_1,...,y_m] = toto(x_1,....,x_n)
.
.
.
où toto est le nom de la fonction, x_1, …, x_n, les n arguments d'entrée et y_1, …, y_m, les m arguments de sortie. Les points verticaux symbolisent les instructions effectuées à l'appel de la fonction.

Le passage des arguments d'entrée dans les fonctions se fait par valeur. Aussi, même si elles sont modifiées dans la fonction les valeurs des paramètres ne sont pas modifiées dans le programme appelant.

Si une des variables de la procédure n'est pas définie à l'intérieur de celle-ci elle doit obligatoirement être fournie en argument d'entrée

La récupération des valeurs calculées par la fonction se fait par les paramètres de sortie ([y_1, …, y_n] dans la définition de toto ci-dessus).

Prenons l'exemple d'une fonction angle, qui doit calculer l'angle formé par le segment d'extrémités (0,0) et (x,y) avec l'horizontale et aussi le dessiner. On peut créer le fichier angle.m contenant les lignes:

```
function [s]=angle(x,y)
s=180*atan(y/x)/pi;
patch([x,0,x],[0,0,y],'y')
axis equal
```

Dans la fenêtre de commandes, on tape
angle(4,5)

Le nom du fichier contenant la fonction porte obligatoirement le nom de cette dernière. On peut mettre plusieurs fonctions dans le même m-file mais seule la fonction du même nom que le fichier peut être utilisée, appelée, à partir de la fenêtre de commandes ou d'une autre fonction ou d'un script. Les autres fonctions éventuellement stockées dans le fichier peuvent s'appeler entre elles mais ne sont pas visibles de l'extérieur.

S'il n'y a qu'un résultat comme dans l'exemple de la fonction angle, on peut se dispenser de le récupérer dans une variable. En revanche s'il y a plus d'un paramètre de sortie, il faut récupérer leurs valeurs dans des variables dans le script d'appel. Regardons par

exemple l'utilisation de la fonction polaire définie ci-dessous et sauvée dans le fichier polaire.m

```
function [r,theta]=polaire(x,y)
r=sqrt(x^2+y^2);
theta=atan(y/x);
```

Pour l'utiliser à partir de la fenêtre Matlab, on tape les instructions suivantes

```
»r=polaire(2,3)    % ou bien
»polaire(2,3)      % Si seulement, le rayon nous intéresse
 ans  =
    3.6055513
»[r,t]=polaire(2,3)  %Si on veut récupérer à la fois le rayon et l'angle
 t  =
     .9828
 r  =
    3.6056
```

Inline functions

Une fonction ne comportant qu'un petit nombre d'instructions peut être définie directement dans la fenêtre de commandes de la manière suivante

```
»angle=inline('180*atan(y/x)/pi')
angle =
     Inline function:
     angle(x,y) = atan(y/x)
»angle(5,4)
 ans  =
    0.6747
```

Les arguments de la fonction angle sont normalement fournis à l'appel dans l'ordre d'apparition dans la définition de la fonction. On peut aussi spécifier les arguments d'appel explicitement

```
»f = inline('sin(alpha*(x+y))','x','y','alpha')
f =
     Inline function:
     f(x,y,alpha) = sin(alpha*(x+y))
»f(0.2,0.3,pi)
ans =
     1
```

Fonctions outils

Enfin pour clore ce paragraphe d'introduction à la programmation sous Matlab, notez que certaines commandes spéciales (voir tableau 2) ne peuvent s'utiliser qu'en relation à une fonction:
nargin, donne le nombre d'arguments d'entrée passés à l'appel de la fonction.

```
function c = testarg1(a,b)
if (nargin == 1)
   c = 2*a;
elseif (nargin == 2)
   c = a + b;
end
```

nargin peut aussi être utilisée pour connaître le nombre prévu d'arguments d'entrée

```
» nargin('testarg1')
ans =
    2
```

La commande nargout fonctionne de manière analogue pour les arguments de sortie.

Tableau 2: Commandes de contrôle.

Instruction	Description
nargin	nombre d'arguments d'entrée d'une fonction
nargout	nombre d'arguments de sortie d'une fonction
error	interrompt l'exécution de la fonction, affiche le message d'erreur et retourne dans le programme appelant.
warning	imprime le message mais ne retourne pas dans le programme appelant
pause	interrompt l'exécution jusqu'à ce que l'utilisateur tape un return
pause(n)	interrompt l'exécution pendant n secondes.
pause off	indique que les pause rencontrées ultérieurement doivent être ignorées, ce qui permet
	de faire tourner tous seuls des scripts requièrant normalement l'intervention de l'utilisateur.
break	sort d'une boucle while ou for.
return	retourne dans le programme appelant sans aller jusqu'à la fin de la fonction.

Algorithmes préprogrammés

Il existe de nombreux algorithmes préprogrammés dans Matlab pouvant être utilisés dans des programmes de simulation plus complexes comme ``boîte noire'' (voir tableau 3). Tous sont répertoriés et présentés dans l'aide en ligne. Leur nom peut se retrouver grâce aux menus search ou index de help).

Tableau 3: Quelques algorithmes préprogrammés

Instruction	Description
fzero(f,a)	recherche des zéros d'une fonction f autour de a
quad(f,a,b)	calcul de l'intégrale d'une fonction f entre a et b
spline(xx,yy)	calcul de la spline cubique passant par les points (xx,yy)
fft(a)	transformation de Fourier rapide du vecteur a
ode23(f,t,y0)	résolution de l'équation y'=f(t,x), y(0)=y0

Les boucles

Il y a deux types de boucles en Matlab : les boucles while et les boucles for. La boucle for parcourt un vecteur d'indices et effectue à chaque pas toutes les instructions délimitées par l'instruction end.

```
»x=1; for k=1:4,x=x*k, end
x =
    1
x =
    2
x =
    6
x =
   24
```

La boucle for peut parcourir un vecteur (ou une matrice) en prenant comme valeur à chaque pas les éléments (ou les colonnes) successifs.

```
»v=[-1 3 0]
v =
   -1    3    0
»x=1; for k=v, x=x+k, end
x =
    0
x =
    3
```

```
x =
     3
```
La boucle while effectue une suite de commandes jusqu'à ce qu'une condition soit satisfaite.
```
»x=1; while x<14,x=x+5,end
x =
     6
x =
    11
x =
    16
```
Les deux types de boucles peuvent être interrompus par l'instruction break . Dans les boucles imbriquées break n'interrompt que la boucle la plus interne.

Tests

Un test est une alternative entre deux commandes (ou groupe de commandes) Matlab sélectionnées suivant le résultat d'une opération logique. Le résultat d'une opération logique (voir tableau 4) est une variable logique ou boléenne qui vaut 1 pour VRAI et 0 pour FAUX.

Dans Matlab on dispose du classique if-else agrémenté du elseif parfois bien utile. La syntaxe est par exemple

```
»x=16
 x  =
    16
»if x>0, y=-x, else y=x, end
 y  =
  - 16
```
On a aussi la possibilité d'utiliser le switch-case, qui a son équivalent dans le langage C
```
switch expression
  case case_expr
    commande,... ,commande
  case case_expr1,case_expr2,case_expr3,...
     commande,... ,commande
...
  otherwise
    commande,... ,commande
end
```
Cet élément de syntaxe permet de gérer plus élégamment les tests portant sur un seul critère pouvant prendre plus de deux valeurs entières, comme dans l'exemple suivant
```
n=round(10*rand(1,1))
switch n
case 0
```

```
   fprintf('cas numero 0')
case 1
   fprintf('cas numero 1')
case 2
   fprintf('cas numero 2')
otherwise
  fprintf('autre cas')
end
```

Tableau 4: Les opérateurs logiques dans les tests

français	test Matlab
et	&
ou	|
non	~
égal	==
différent	~=
plus petit que	<
plus grand que	>
plus petit ou égal à	<=
plus grand ou égal à	>=

Lecture et écriture au clavier et dans des fichiers

On peut avoir à lire des données numériques ou alphanumériques dans un fichier, par exemple les conditions initiales d'un calcul ou un maillage qui a été généré par un autre programme. Inversement, on peut vouloir sauver le résultat d'un calcul, pour s'en servir ultérieurement dans Matlab ou dans un autre programme (voir tableau 5).

M = dlmread('NomDeFichier','Delimiteur')

Lit des données numériques du fichier ASCII NomDeFichier, chaque nombre est séparé du suivant par le caractère Délimiteur ou par un retour à la ligne. La virgule et l'espace sont des délimiteurs par défaut.

[A,B,C,...] = textread('NomDeFichier','format')

Lit les variables avec le format spécifié jusqu'à épuisement du fichier. (Le nombre de variables à lire et le nombre de données dans le fichier doivent correspondre.)

fid=fopen('NomDeFichier') % ouvre le fichier
A = fscanf(fid,format)
[A,count] = fscanf(fid,format,size)
Lit les données avec le format spécifié. Un format est une chaine de caractères spécifiant en lecture le type de données à lire:
'%d' pour un entier
'%f' pour un réel
'%c'pour un caractère.
On peut éventuellement spécifier le nombre maximal de données à lire avec le paramètre size, et le paramètre de sortie count renvoie le nombre de données effectivement lues.

Pour écrire des résultats sur l'écran la méthode la plus simple et de taper le nom de la variable sans la faire suivre d'un point virgule. Si on veut un format plus lisible, en particulier, afficher plusieurs variables sur la même ligne, on peut utiliser la commande fprintf, comme dans l'exemple ci-dessous

```
a=1.5;
b=2;
fprintf('a = %f et b= %d',a,b);
  a =1.5 et b=2
```

Tableau 5: Les commandes d'entrée-sortie et les accès fichiers

dlmread('NomDeFichier','delimiteur')	lecture du fichier
dlmwrite('NomDeFichier',M,'delimiteur')	écriture de M dans le fichier
textread('NomDeFichier','format')	lecture du fichier
fid=open('NomDeFichier')	ouverture du fichier NomDeFichier
[A,count]=fscanf(fid,'format')	lecture du fichier ouvert par open
fprintf(fid,'format',données)	écriture des données avec un format
close(fid)	fermeture
fprintf('format',données)	écriture des données avec un format
fprintf('format',données)	écriture des données avec un format

Représentation graphique

Dans toutes les représentations graphiques, le logiciel se base sur des données discrètes rangées dans des matrices ou des vecteurs colonnes. Par exemple, pour

représenter des courbes du type y=f(x) ou des surfaces z=f(x,y), les données x,y,z doivent être des vecteurs colonnes (x et y) ou des matrices (z) aux dimensions compatibles. L'instruction de dessin correspondante (par exemple plot(x,y) pour tracer des courbes planes) est alors utilisée et éventuellement complétée par des arguments optionnels (couleur, type de trait, échelle sur les axes, ...). La visualisation du résultat s'effectue dans une fenêtre graphique (avec possibilité de zoom, de rotation, d'impression).

Exemple de représentation graphique en dimension deux

En exécutant le script suivant:

```
x=linspace(0,pi,30);   % crée un tableau de 30 composantes uniformément
% réparties entre 0 et pi
y=sin(x);
plot(x,y)    %relie les points (xi,yi) par un trait continu noir
plot(x,y,'p-b')  %relie les points (xi,yi) par un trait continu de couleur et
%matérialise les points avec un symbole
plot(x,y,'pb')  %matérialise les points (xi,yi) avec un symbole de couleur
```

Les points peuvent être matérialisés par le symbole p prenant les valeurs suivants: o.*+x-. Les couleurs sont repérées par leur initiale en anglais: r(ed), b(lue), blac(k), w(hite), y(ellow), m(agenta), g(reen). On peut rajouter un titre à la figure avec la commande title

```
title('sin(x) sur l''intervalle [0,pi]')
```

(Remarquer l'emploi d'une double apostrophe pour en faire figurer une dans une chaîne de caractères délimitée justement par deux apostrophes.)

On peut représenter plusieurs courbes sur la même figure de plusieurs manières: d'abord par un seul appel à la fonction plot

```
plot(x,cos(x),x,sin(x),x,exp(-x))    % Matlab va automatiquement utiliser des couleurs
%différentes pour chaque courbe
plot(x,cos(x),'o-r',x,sin(x),'x-b',x,exp(-x),'*-g')    % pour spécifier le type
% de symbole et la couleur à utiliser pour chaque courbe
legend('cos(x)','sin(x)','exp(-x)')  % pour rajouter une légende
```

Par défaut la fenêtre graphique est effacée avant chaque commande plot. Pour superposer des courbes par des appels successifs à cette fonction, il faut auparavant avoir utilisé la commande hold on.

Tableau 6: Principales instructions graphiques.

Instruction	Description
plot(x,y)	tracé de la courbe passant par les points (x,y)
loglog(x,y)	idem avec échelle logarithmique sur les deux axes

semilogx(x,y)	idem avec échelle logarithmique sur l'axe Ox
semilogy(x,y)	idem avec échelle logarithmique sur l'axe Oy
plotyy(x,y,x,z)	courbe (x,y) avec l'axe 0y à gauche,
	et courbe (x,z) avec l'axe 0z à droite
xlabel('label')	légende pour l'axe 0x
ylabel('label')	légende pour l'axe 0y
title('label')	titre au dessus du graphique
legend('lab1','lab2','lab3',...)	légende avec une chaîne de caractères pour chaque courbe
text(x,y,'label')	chaîne de caractères à la position x,y
plot3(x,y,z)	tracé de la surface passant par les points (x,y,z)
hold on, hold off	active/désactive la conservation de la fenêtre graphique à l'appel de la fonction plot

Autres types de représentation

Outre la représentation cartésienne de courbes ou de surfaces, il existe d'autres possibilités pour illustrer graphiquement un résultat. On peut citer parmi les plus utiles, les instructions contour, ezmesh (pour tracer les courbes de niveau d'une surface paramétrique), mesh, ezplot3 (courbes paramétriques dans l'espace), hist, rose (histogramme d'un échantillon de données statistiques),...